MINDS at SemEval-2025 Task 8: Question Answering Over Tabular Data via Large Language Model-generated SQL Queries

Flavio Giobergia
Politecnico di Torino
Turin, Italy
flavio.giobergia@polito.it

Abstract

The growing capabilities of Large Language Models (LLMs) have opened up new opportunities for answering questions based on structured data. However, LLMs often struggle to directly handle tabular data and provide accurate, grounded answers. This paper addresses the challenge of Question Answering (QA) over tabular data, specifically in the context of SemEval-2025 Task 8. We propose an LLM-based pipeline that generates SQL queries to extract answers from tabular datasets. Our system leverages In-Context Learning to produce queries, which are then executed on structured tables, to produce the final answers. We demonstrate that our solution performs effectively in a few-shot setup and scales well across tables of different sizes. Additionally, we conduct a data-driven error analysis to highlight scenarios where the model encounters difficulties. We make the code available at https://github.com/fgiobergia/SemEval2025-Task8.

1 Introduction

The introduction of general purpose Large Language Models has made it possible to address a wide variety of tasks, with no need to fine-tune a specific model every time. This capability has enabled a widespread adoption of LLMs to address a wide variety of tasks (e.g., machine translation (Xu et al., 2024), document classification (Giobergia et al., 2024) and summarization (Pu et al., 2023)). However, these models often provide answers that are either based on the data available in the training data (i.e., they cannot leverage external data), or that are not grounded in factual data – a phenomenon referred to as hallucinations. This can lead to issues like generating inaccurate facts, fabricating citations, or incorrect summarizations despite the model seeming confident in its output.

Although attempts have been made to detect and mitigate hallucinations (Ji et al., 2023; Borra et al., 2024), off-the-shelf models used for In-Context Learning still cannot provide meaningful answers to questions related to a specific data source, unless access to the data source itself is provided. One such example is the answering of questions that can only be addressed based on the contents of a separate data source. The SemEval 2025 Task 8 – Question Answering Over Tabular Data (Osés Grijalba et al., 2025) addresses exactly this kind of scenario, by framing a Question Answering (QA) problem, with answers that can be extracted from tabular data. In this paper we address the challenge by building a LLM-based pipeline that receives information on the structure of the target table, produces a SQL query to answer the question, and provides a final answer by executing the query on the tabular data. We show that the proposed solution achieves remarkable results in few-shot mode, and that it provides satisfactory performance regardless of table size. We additionally perform a data-driven error analysis to identify corner cases that the LLM cannot easily address; and improve model performance by manually labelling useful cases to be used as shots within the prompt.

The rest of the paper is organized as follows: Section 2 introduces the task, dataset and metrics. Section 3 presents the main methodology adopted, with an overview of the pipeline, as well as the error analysis that has been conducted to identify promising shots to be used. Section 4 presents the main results obtained. Finally, Section 5 wraps up the paper, with considerations and possible future extensions of the work.

2 Problem description

SemEval Task 8 consists in addressing QA problems based on tabular data. To this end, the Task makes use of DataBench (Grijalba et al., 2024). DataBench is the first benchmark that makes use of real-world tables, with a wide variety of distinct questions related to various data types. Answers to the questions are in the form of either a number, a

Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025), pages 2219–2224

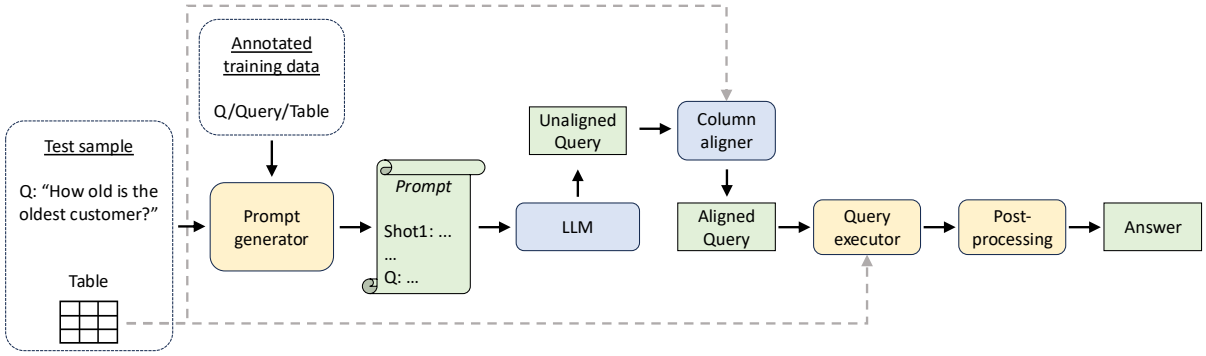

Figure 1: Overview of the architecture for the proposed solution. In green are the results produced (intermediate, e.g. queries, and final, i.e. the answer). In blue are the LM-based components of the solution. In yellow are the pre- and post-processing steps.

categorical value, a boolean value or lists of several types. DataBench includes a training set (49 tables, 988 questions), a validation set (16 tables, 320 questions) and a test set, specifically released for SemEval (15 tables, 522 questions).

The challenge consists of two subtasks: a *full* one, where the entire table (all rows, all columns) are used, and a *lite* one, where only a subset of all rows and columns are used.

The quality of the provided answers is quantified in terms of the fraction of correctly answered questions (i.e., accuracy).

3 Proposed methodology

We present the architecture of the overall solution in Figure 1. The architecture takes any one question (and related table) and produces the prompt with the *prompt generator*. Next, this prompt is used to generate an answer query from an LLM. This *unaligned query* may contain some errors that are adjusted by a *column aligner* module. The correct query is then automatically executed on the target SQLite table. The answer returned by the query is post-processed to produce the final answer. The rest of this section presents, in more details, the various blocks.

3.1 Prompt generator

The prompt generator is used to produce the prompt for the LLM. The prompt is comprised of three parts: (1) the general description of the task to be addressed (i.e., producing the query that answers a question), (2) some examples (shots) of questions, summaries of tables, and queries that are expected as the answer, and (3) the actual question and table summary to be addressed by the LLM.

```
The following are examples of
question/table/answer.
Provide the answer to the last
question. Only include the query.
Use exactly the specified column names,
as reported between backticks ``.
If the answer is boolean, use CASE WHEN
... THEN ... ELSE ... END.

Question: [Question for shot 1]
Table (`column name`: list of values):
`column1`: value1, value2, value3, ...
`column2`: value4, value5, value6, ...
`column3`: value7, value8, value9, ...
...
Answer: [Query for shot 1]

[shot 2]
...
Question: [Test question]
Table (`column name`: list of values):
[Test table]
Answer:
```

Listing 1: Example of a prompt used in our experiments. In blue is the problem description, in orange the shots, in green the actual question.

We report in Listing 1 a summarized prompt. As shown, each shot is represented as a triplet (Question, Table, Answer). The *Question* is the natural language question that needs to be addressed. The *Table* is a list of columns found in the table of interest. For each column, in addition to the column name, a few (5, in our case) sample values are reported. The *Answer* is the query that can be executed on the specified table to extract the correct answer. We note that the training set does not contain queries associated to the actual answers. We discuss the annotation process of a limited number of queries as a part of Section 3.5. The final question is appended at the end of the list of shots. Because of memory limitations, we limit the number of shots used in the prompt to 7.

Finally, we note that the usage of the `CASE ... WHEN ... THEN ... ELSE ... END` allows the model to directly return true/false answers, instead of 0/1. This is convenient to simplify the post-processing step. We empirically verified that the models adopt the pattern correctly if explicitly asked to do so in the prompt, and if the shots contain such examples.

3.2 Large Language Model

The key component for the proposed solution is an LLM, that produces the query to address the answer. We considered small and medium LLMs that have been specifically fine-tuned on code generation tasks, and that have been instruction-tuned. More specifically, we identified the two most promising candidates in Code Llama 7b and 34b (Roziere et al., 2023), Codestral 22b (MistralAI, 2024) and Qwen2.5-Coder 32b (Hui et al., 2024). The experimental section contains a comparison between the two models.

The output of the model is, in general, a valid query that can be executed. However, a further alignment step is required, in some cases, to guarantee that the columns adopted in the query actually exist.

3.3 Column aligner

We note that, when generating the query, the LLM sometimes struggles to correctly specify some of the column names; despite receiving the correct names as a part of the prompt. To fix this problem, we introduce a step of semantic alignment, to replace invalid generated columns with correct ones.

This problem likely stems from the fact that the model struggles to use the correct names, if they contain non-SQL-standard characters. For instance, in dataset `054_Joe`, one of the attribute names is `author_name<gx:category>`. This attribute is always used as `author_name` in the LLM-generated queries, resulting in the execution of invalid queries.

Since the model wraps all column names with backticks ``, we can easily extract the set C_Q of columns used in the generated queries. The set C_V of valid column names is also known from the table schema. If $C_Q \setminus C_V \neq \emptyset$, some columns adopted in the query are not part of the available columns. Only for those columns, we apply an alignment step based on semantic similarity.

We adopt a pre-trained encoder-only transformer model (e.g., BERT (Devlin et al., 2019)) to produce vector encodings for the columns in $C_Q \setminus C_V$ (the columns to be replaced), and for columns in C_V (the columns to be used for the replacements). We replace each invalid column with the most similar valid column (based on cosine similarity). We acknowledge that using BERT for the encoding of column names (i.e., short strings of text, without adequate context) goes against the rationale of the model. Alternative approaches (e.g., fastText (Mikolov et al., 2018)) also allow generating vector representations for words, without requiring a context. This is also true for representations across multiple languages (Grave et al., 2018) and tasks (e.g., sentiment analysis (Giobergia et al., 2020)), even for out-of-vocabulary words (Savelli and Giobergia, 2024): both properties could be useful, when handling column names. However, we empirically observe satisfactory results even for BERT, and use it despite the lack of a context.

3.4 Query executor and post-processing

The Parquet datasets are converted into SQLite tables, which are then used to run the SQL queries generated by the LLM. This step can produce an error, if a query is not valid. In this work, we do not include iterative steps to improve the quality of the results. However, it is reasonable to assume that the results could benefit from it. If the query executes correctly, the result is adjusted during a final, very simple post-processing step, to produce the final answer (for instance, length-1 lists of values are returned as a single value).

Medoid question	Closest Train question
What are the bottom 2 languages in terms of tweet count?	Which are the top 4 events with the highest average number of comments?
Has the author with the highest number of followers ever been verified?	What is the maximum number of reviews a property has received?
Identify the top 3 foods with the least amount of sugar.	Identify the 3 departments with the lowest average satisfaction levels.
How many respondents are from the region adjacent to the South Atlantic Ocean?	How many participants are from the United Kingdom?

Table 1: Questions for the 4 clusters (as defined by their medoids), with corresponding most similar questions (via cosine similarity) in the training set. The questions in the training set (and corresponding manually annotated queries) have been included as a part of the prompt.

3.5 Error analysis

We initially annotated a small number of questions with the corresponding SQL query. This pool of annotations provides the intial shots used in the definition of the prompts. We then run the proposed pipeline, on the validation set. We isolate the cases that produce incorrect answers, and try to identify the most recurring types of questions that cannot be addressed by the LLM. To identify common patterns in incorrectly answered questions, we use clustering. We first build a vector representation for each incorrectly answered question, using an encoder-only model (e.g., RoBERTa large (Liu et al., 2019)). Then, we apply K-medoids (Park and Jun, 2009) to identify 4 clusters[1] of questions, with the corresponding medoids (i.e., the most representative question, for each cluster). Finally, we identify, among the training questions, the most semantically similar ones[2] to each of the medoids. We manually annotate those 4 questions with the corresponding queries, which are then included as a part of the prompt. We show, in Table 1, the medoids (from the validation set), paired with the corresponding most similar training question.

[1]The number of clusters has been selected based the total number of shots that could be included in the prompt.

[2]Based on the cosine similarity computed on the latent vectors.

Model	Accuracy (full)	Accuracy (lite)
Code Llama 7b	49.81	52.87
Code Llama 34b*	6.70	60.34
Codestral v0.1 22b*	**72.41**	**74.14**
Qwen2.5-Coder 32b*	<u>72.03</u>	<u>69.92</u>

Table 2: Performance, in terms of accuracy, on the full and lite tasks, using three different LLMs. (*) represents models that have been quantized with PTQ, on 4 bits. Best result in **bold**, second best <u>underlined</u>.

4 Experimental results

We present the main results obtained using the proposed pipeline. First, we evaluate the performance for different LLMs. Next, we highlight some of the limitations in the responses generated by the different models.

4.1 Main results

We test four separate instruction-tuned LLMs, trained on code generation tasks: Code Llama 7b, Code Llama 34b (Roziere et al., 2023), Codestral v0.1. 22b (MistralAI, 2024) and Qwen2.5-Coder 32b (Hui et al., 2024).

We quantized CodeLlama 34b, Codestral 22b and Qwen2.5-Coder 32b with Post-Training Quantization (PTQ) on 4 bits, to execute them on the available hardware.

Table 2 presents the main results obtained on the full and lite tasks, in terms of accuracy, using the proposed solution.

Interstingly, Codestral emerges as the clear winner, on both tasks, closely followed by Qwen2.5-Coder. Interestingly, in the Code Llama family, the 7b version obtains more consistent performance across the tasks, whereas the 34b version obtains abysmally low performance on one task, and reasonable results on the other – despite the minor differences between the tasks.

4.2 Common mistakes

We note that the proposed approach sometimes produces SQL queries that cannot be executed. For such cases, the answer to the question is left blank. Table 3 reports the number of blank answers given by the four models under study.

This result helps explain the poor performance obtained by Code Llama 34b, especially for the full task: a large number of answers is in the form of queries that do not execute correctly. Upon manual inspection, we note that Code Llama 34b often

Model	% missing (full)	% missing (lite)
Code Llama 7b	**7.47**	**7.85**
Code Llama 34b*	18.42	10.34
Codestral v0.1 22b*	<u>7.85</u>	<u>8.05</u>
Qwen2.5-Coder 32b*	12.26	15.71

Table 3: Fraction of missed queries, for each model (i.e., queries that did not correctly execute). (*) represents models that have been quantized with PTQ, on 4 bits. Best result in **bold**, second best <u>underlined</u>.

returns answers in natural language that do not include the requested query (e.g., by attempting to answer the question directly, or providing irrelevant comments). Other models do not behave as poorly, with consistent behaviors between the two tasks. Qwen2.5-Coder, interestingly, produces a large number of empty results[3]. Despite this high percentage of blank answers, it is impressive that Qwen2.5-Coder achieved the second-best performance: we argue that, if this behavior was to be limited (e.g., with sufficient prompt engineering, which has not been carried out in this work), Qwen could prove to be an even more competitive solution.

We highlight an additional problem often found in incorrect answer for some of the models. This common error consists in returning a list of values when a single, aggregate one was expected. This is the case, for example, with questions that require an answer on the overall behavior across the dataset. For instance, the question *are all employees older than 20?* has a single true/false outcome. The question should be addressed using a query such as `SELECT COUNT(*) = (CASE WHEN age > 20 THEN 1 END) FROM data`. Instead, models often incorrectly produce a query that returns an outcome for each entry in the table, such as `SELECT age > 20 FROM data`.

Answers obtained from these queries are generally much longer than valid answers. Based on this consideration, Figure 2 presents the distribution in answer lengths for the models considered, on the *full* task. Longer answers (300+ characters) are generally linked with incorrect results. Interestingly, all models but Qwen2.5-Coder present several of these failure cases. In particular, Code Llama 7b is affected by this problem the most.

We attempted to mitigate this problem by introducing specific shots that provide correct queries. However, those shots only marginally helped: as soon as a slightly different request was encountered, the LLM reverted to the undesired behavior.

[3]We additionally note that we failed to test the 7b version, as it consistently produced invalid responses.

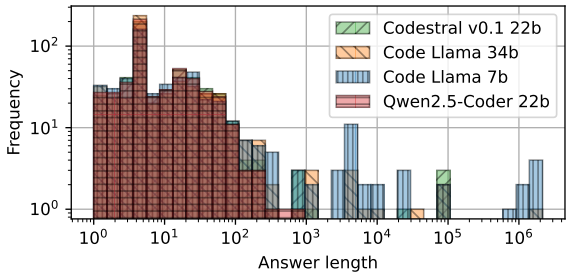

Figure 2: Distribution of the length of the answers obtained by executing the queries generated by different models. Lengths above 2-300 characters can generally be considered as the result of faulty queries.

5 Discussion and conclusions

In this paper we presented a solution to the QA problem on tabular data from SemEval 2025 Task 8. The solution is based on generating SQL queries that are executed to produce answers to the proposed questions. We make use of an instruction-tuned LLM trained for code generation. We show that the pipeline allows to achieve acceptable performance. We tested several models and find Codestral v0.1 22b to be the one providing the most accurate results. Interestingly, almost all considered models show no gap in performance between the full and the lite versions of the task, indicating robustness to noisy information (e.g., the presence of unused columns). We acknowledge that one of the main limitations of the proposed approach is the conversion of the datasets into SQLite tables: the original Parquet datasets contain potentially complex data types (e.g., lists of values), which cannot be easily cast into SQLite columns. As such, we expect that a transposition of the proposed solution to other querying approaches (e.g., Python-based) may yield even more promising results.

Acknowledgements

This study was carried out within the FAIR - Future Artificial Intelligence Research and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1555 11/10/2022, PE00000013). This manuscript re-

flects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

References

Federico Borra, Claudio Savelli, Giacomo Rosso, Alkis Koudounas, and Flavio Giobergia. 2024. MALTO at SemEval-2024 task 6: Leveraging synthetic data for LLM hallucination detection. In *Proceedings of the 18th International Workshop on Semantic Evaluation (SemEval-2024)*, pages 1678–1684, Mexico City, Mexico. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Flavio Giobergia, Luca Cagliero, Paolo Garza, and Elena Baralis. 2020. Cross-lingual propagation of sentiment information based on bilingual vector space alignment.

Flavio Giobergia, Alkis Koudounas, and Elena Baralis. 2024. Large language models-aided literature reviews: A study on few-shot relevance classification. In *2024 IEEE 18th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–5. IEEE.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Jorge Osés Grijalba, Luis Alfonso Ureña-López, Eugenio Martínez Cámara, and Jose Camacho-Collados. 2024. Question answering over tabular data with databench: A large-scale empirical evaluation of llms. In *Proceedings of LREC-COLING 2024*, Turin, Italy.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

MistralAI. 2024. Codestral. Accessed: 2025-02-28.

Jorge Osés Grijalba, L. Alfonso Ureña-López, Eugenio Martínez Cámara, and Jose Camacho-Collados. 2025. Semeval-2025 task 8: Question answering over tabular data. In *Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025)*, pages 1015–1022, Vienna, Austria. Association for Computational Linguistics.

Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341.

Xiao Pu, Mingqi Gao, and Xiaojun Wan. 2023. Summarization is (almost) dead. *arXiv preprint arXiv:2309.09558*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Claudio Savelli and Flavio Giobergia. 2024. Enhancing cross-lingual word embeddings: Aligned subword vectors for out-of-vocabulary terms in fasttext. In *2024 IEEE 18th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–6. IEEE.

Haoran Xu, Amr Sharaf, Yunmo Chen, Weiting Tan, Lingfeng Shen, Benjamin Van Durme, Kenton Murray, and Young Jin Kim. 2024. Contrastive preference optimization: Pushing the boundaries of llm performance in machine translation. *arXiv preprint arXiv:2401.08417*.